MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
SEÇÃO DE ENGENHARIA DE COMPUTAÇÃO

ANTÔNIO LUÍS SOMBRA DE MEDEIROS
LUIS HELDER LIMA BARBOSA

ANÁLISE DE SEGURANÇA NA PLATAFORMA ANDROID

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA

ANTÔNIO LUÍS SOMBRA DE MEDEIROS
LUIS HELDER LIMA BARBOSA

ANÁLISE DE SEGURANÇA NA PLATAFORMA ANDROID

Iniciação à Pesquisa apresentada ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia.

Orientador: Anderson Fernandes P. dos Santos, D.Sc.

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

005.268 Medeiros, Antônio Luís Sombra de
M488s Análise de Segurança na Plataforma Android / Antônio Luís Sombra de Medeiros, Luis Helder Lima Barbosa. - Rio de Janeiro: Instituto Militar de Engenharia, 2014.

47p.: il.

Iniciação à Pesquisa (IP) - Instituto Militar de Engenharia - Rio de Janeiro, 2014.

1. Engenharia de computação - Iniciação à pesquisa. 2. Plataformas móveis - Processamento de dados. 3. Segurança da informação. I. Barbosa, Luis Helder Lima. II. Santos, Anderson Fernandes P. dos.. III. Título. IV. Instituto Militar de Engenharia.

INSTITUTO MILITAR DE ENGENHARIA

ANTÔNIO LUÍS SOMBRA DE MEDEIROS
LUIS HELDER LIMA BARBOSA

ANÁLISE DE SEGURANÇA NA PLATAFORMA ANDROID

Iniciação à Pesquisa apresentada ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia.

Orientador: Anderson Fernandes P. dos Santos, D.Sc.

Aprovada em 13 de Junho de 2014 pela seguinte Banca Examinadora:

Anderson Fernandes P. dos Santos, D.Sc., do IME

Sérgio dos Santos Cardoso Silva, M.Sc., do IME

Julio Cesar Duarte, D.C., do IME

Rio de Janeiro
2014

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API *Application Programming Interface*

HTTP *Hyper Text Transfer Protocol*

HTTPS *Hyper Text Transfer Protocol Secure*

IME Instituto Militar de Engenharia

IMEI *International Mobile Station Equipment Identity*

JIT *Just in Time*

MCS *Multi Categories Security*

MLS *Multi Level Security*

OpenGL *Open Graphics Library*

P2P *Peer to Peer*

SDK *Software Development Kit*

SHA *Secure Hash Algorithm*

SIP *Session Initiation Protocol*

SMS *Short Message Service*

SSL *Secure Sockets Layer*

VoIP *Voice over IP*

VPN *Virtual Private Network*

RESUMO

O Android assumiu uma posição de destaque no meio tecnológico e já é o sistema operacional mais utilizado em dispositivos móveis ao redor do mundo, ocupando também a segunda colocação entre os sistemas operacionais em geral, atrás apenas do Windows. Esse grande crescimento trouxe também problemas para a plataforma e seus usuários, visto que ela passou a ser mais visada por *crackers*, tendo sido descobertas várias vulnerabilidades a cada versão lançada. Com essa visibilidade, a segurança durante o uso do Android tornou-se um ponto importante a ser levado em consideração por parte dos desenvolvedores e usuários da plataforma.

Neste trabalho fez-se uma descrição sobre os principais elementos que compõem a plataforma Android, como a máquina virtual Dalvik e o *kernel*, e de como eles interagem entre si. Além disso, foram analisados pontos de vulnerabilidade do sistema, bem como mecanismos de segurança incluídos no Android, como o SELinux (*Security Enhanced Linux*) e o sistema de permissões. Por fim foi feita uma comparação entre o Android e outras plataformas móveis dos principais fatores que influenciam na segurança de cada sistema, além de serem sugeridos possíveis pontos de melhoria na segurança do Android como um todo.

O presente estudo descreve o funcionamento geral da plataforma Android, bem como os principais mecanismos que o Android emprega para evitar que os dispositivos sejam suscetíveis a ataques cibernéticos, além de explicitar vulnerabilidades existentes no sistema e como o mesmo poderia ser melhorado em alguns aspectos.

ABSTRACT

Android has taken an important position in the technology market and is already the most used mobile operational system in the world, also standing at second position among operational systems in general, only behind Windows. This great growing in use has also brought some problems to the platform and its users, as it becomes more and more targeted by crackers, and many vulnerabilities are discovered as new versions are released. With such visibility, using Android safely became an important concern between its developers and users.

In this paper, the main components of the Android platform are described, including the Dalvik virtual machine and the kernel, and how they interact with each other. Besides, the vulnerability spots in the system were analyzed, along with the security mechanisms included in Android, like SELinux (Security Enhanced Linux) and the permissions system. Lastly, it was done a comparison between Android and other mobile platforms, regarding the main factors that influence security in each system, in addition to some suggestions that could improve the security in Android as a hole.

This study describes how the Android platform works in general, as well as the main mechanisms that Android uses to prevent its devices from bring exploited. Besides, it highlights weaknesses still present in the system and how the platform could be made better in some aspects.

1 INTRODUÇÃO

Falhas de segurança são um problema que acompanha o desenvolvimento de *software* desde seu início. Elas são causadas pelos mais variados fatores, seja pelo despreparo dos programadores ou pela necessidade das empresas lançarem produtos rapidamente para fazer frente à competição, o que faz com que sejam lançados novos *softwares* sem que os devidos testes de segurança tenham sido realizados. Estas falhas são usadas por *crackers* para atacar pessoas, empresas e governos e roubar informações.

À medida que cresce o número de dispositivos e, consequentemente, de *softwares* usados, cresce também a preocupação com tais ataques, pois o dano que uma falha pode causar aumenta proporcionalmente ao número de pessoas usando programas que contenham essa falha. Além disso, programas largamente utilizados tendem a ser o alvo principal de indivíduos mal intencionados, visto que elas normalmente desejam atingir o maior número de pessoas possível, excetuando-se casos em que os ataques são direcionados a alvos críticos específicos, como uma usina nuclear, por exemplo.

1.1 MOTIVAÇÃO

A computação móvel é uma área que surgiu há pouco tempo e, apesar do rápido crescimento, ainda há muito espaço para inovações. O Android em particular tem se mostrado um sistema operacional móvel extremamente competitivo. Tendo superado seus concorrentes em número de usuários, já responde por mais 79% dos usuários de *smartphones*[1] e mais de 62% dos usuários de *tablets*[2].

Além disso, a segurança da informação, um tema que sempre mereceu lugar de destaque no mundo da informática, também vem crescendo em importância, devido à popularização da tecnologia, que levou ao aumento do números de ataques cibernéticos, e também devido às recentes denúncias de espionagem entre países.

No contexto do IME, a segurança da informação também é de grande interesse, visto que uma das frentes do Exército Brasileiro é a segurança cibernética. Há também a possibilidade de que a pesquisa seja extendida futuramente no instituto, fazendo-se um estudo da segunraça do Android em outros dispositivos, por exemplo, ou testando-se a eficácia das proteções ou dos ataques aqui apresentados.

1.2 OBJETIVOS

Este trabalho tem por objetivo fazer um estudo sobre a segurança no Android e nos aplicativos feitos para a plataforma.

Será feito primeiramente um estudo do funcionamento geral da plataforma, de forma a estabelecer uma base de conhecimento. Será apresentada uma pesquisa das vulnerabilidades mais importantes já encontradas e uma análise sobre possíveis fatores que podem estar contribuindo para o aumento ou diminuição da segurança, assim como um levantamento de semelhanças e diferenças com outras plataformas móveis com relação à segurança, em termos estatísticos e funcionais. Além disso, serão sugeridas melhorias que podem ser adotadas para incrementar a segurança no uso do sistema Android.

1.3 JUSTIFICATIVA

Os dispositivos móveis têm sofrido um aumento significativo no seu uso e já são uma realidade na vida de bilhões de pessoas ao redor do mundo. Eles são usados para os mais variados propósitos, entre os quais estão: envio de *e-mails*, redes sociais, compras, transações financeiras e entretenimento. A mobilidade tornou mais prática a realização dessas atividades e o Android tem sido o sistema preferido pela maioria dos usuários de computação móvel.

Com o crescente uso do Android nos últimos anos, cada vez mais *crackers* têm voltado sua atenção para a plataforma. Isso se reflete no crescente número de *malwares* que têm surgido para atormentar os usuários do sistema. De acordo com o relatório anual de segurança da Cisco[3], 99% de todos os *malwares* identificados em 2013 tiveram o sistema Android como alvo. O mesmo relatório também reporta que há uma demanda mundial não atendida de mais de uma milhão de profissionais de segurança.

Nesse cenário, se faz cada vez mais importante que os desenvolvedores de aplicativos tenham acesso a informações que os auxiliem a tornar seus aplicativos mais seguros. Por isso é importante que sejam feitos estudos sobre a segurança na plataforma Android e que suas vulnerabilidades sejam encontradas e corrigidas, antes que possam ser exploradas.

1.4 METODOLOGIA

A primeira etapa na realização desta pesquisa consistiu na coleta e estudo de fontes bibliográficas, incluindo artigos e *sites*, sobre o funcionamento do Android e sobre os aspectos de segurança da plataforma. Com o material em mãos, iniciou-se a escrita de fato

do texto, com foco em explicitar o funcionamento dos principais elementos constituintes da plataforma e como eles se integram para gerar a experiência final percebida pelo usuário.

Na etapa seguinte, foi feita uma explanação sobre os principais mecanismos do Android que visam à proteção do sistema e do usuário final, além de métodos de ataque relevantes e vulnerabilidades descobertas ao longo do tempo.

Para concluir, foi feita uma comparação entre o Android e outras plataformas móveis no que tange à segurança, além de terem sido sugeridas melhorias que, se implementadas no sistema, poderiam auxiliar nos esforços de aprimorar a confiabilidade do Android.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

No capítulo 2 é apresentado um pouco da história do Android e de suas versões. Em seguida são apresentados os principais métodos que o sistema usa para executar e gerenciar as aplicações. Após isso, é apresentada a *Dalvik VM*, máquina virtual responsável pela execução dos aplicativos escritos em Java, além de seus principais recursos. Posteriormente, faz-se uma breve apresentação dos recursos oferecidos pelo *kernel* do Android.

No capítulo 3 é feita uma apresentação das principais ferramentas de proteção do Android. Começando pelos mecanismos de controle de acesso presentes no *kernel*, que são a *Sandbox* e o SELinux, é explicada também uma ferramenta para verificação do sistema de arquivos, a *dm-verity*, que também funciona a nível de *kernel*. Num nível mais próximo do usuário, discute-se o sistema de permissões do Android, o mecanismo de assinatura de aplicativos e as ferramentas para verificação de aplicativos, o *Verify Apps* e o *Bouncer*.

No capítulo 4 são discutidas algumas vulnerabilidades presentes na plataforma Android. É explicado o conceito de escalada de privilégio, excesso de permissões, entre outros, assim como são apresentados exemplos de casos de vulnerabilidade gerados por API's inseguras e como ocorrem e atuam aplicações maliciosas nos *smartphones* com Android.

No capítulo 5 foram sugeridas possíveis melhorias para a plataforma Android, que poderiam auxiliar nos esforços pela segurança do sistema.

No capítulo 6 foram feitas comparações entre as plataformas móveis atuais mais utilizadas: Android, iOS e Windows Phone. Foram destacados pontos relacionados à segurança destas plataformas.

2 PLATAFORMA ANDROID

O Android é um sistema operacional e plataforma de código aberto para dispositivos móveis desenvolvido pela Google e a *Open Handset Alliance*[4], uma aliança atualmente formada por 84 empresas, incluindo a Google e empresas líderes no setor tecnológico. Seu código-fonte está disponível para consulta no repositório central[5].

Os recursos que o Android possui atualmente são diversos, suportando mídias como áudio, vídeo e diversos formatos de imagens, *bluetooth*, 3G, *wi-fi*, câmera, acelerômetros e gráficos otimizados por bibliotecas de gráficos 2D e 3D.

2.1 HISTÓRIA

Em 2003 a Android Inc foi criada, em Palo Alto (Califórnia-EUA), sendo comprada pela Google em Julho 2005[6].

Em 5 de Novembro de 2007 a Google anunciou o nascimento do Android como uma plataforma móvel baseada no *kernel* Linux versão 2.6 e a criação da *Open Handset Alliance*, sendo uma junção da Google e mais 33 empresas parceiras na época, como LG, Samsung e Intel[7].

Em 21 de Outubro de 2008 o Android se tornou *open source*[8]. Um dia depois foi lançado o primeiro aparelho celular com Android, o HTC G1.

Segue uma breve histórico das principais versões do Android com suas características marcantes:

- 1.6 *Donut* (API level 4) - Lançada em 15 de Setembro de 2009, baseada no *kernel* 2.6.29[9], trouxe um sistema de busca otimizado, que permite ao usuário buscar no histórico, contatos ou na própria Internet através de uma caixa de pesquisa na própia tela inicial[10].

- 2.0-2.1 *Eclair* (API level 5-7) - Lançada em 26 de Outubro de 2009 a versão 2.0, baseada no *kernel* Linux 2.6.29[9], trouxe suporte para HTML5, facilidades para gerenciamento de contatos e novos recursos para a câmera, trazendo APIs com suporte para *flash*, que possibilitam controle de zoom e a instalação de aplicativos no cartão SD. As demais versões do *Eclair* vieram entre Dezembro e Janeiro de 2010, corrigindo erros e trazendo pequenas mudanças nas APIs[11].

- 2.2-2.2.3 *Froyo* (API level 8) - Lançada em 20 de Maio de 2010, baseada no *kernel* Linux 2.6.32[9], a versão 2.2 teve como principal característica a inserção do compilador JIT na arquitetura, aumentando a performance em relação a versões anteriores[12].

- 2.3-2.3.7 *Gingerbread* (API level 9-10) - Lançada em 6 de Dezembro de 2010, baseada no *kernel* Linux 2.3.35[13], a versão 2.3 Gingerbread trouxe uma interface com o usuário (UI) melhorada, suporte para telefonia SIP e VoIP, e otimizações para jogos, com a introdução do *garbage collector* concorrente. Trouxe APIs que possibilitam aplicativos acessarem qualquer câmera do celular, gerenciar dispositivos de áudio e criar texturas e superfícies diretamente de código nativos, entre outras utilizações de código nativo para diminuir o processamento. As demais versões até a 2.3.7 trouxeram várias correções de APIs e *bugs*.

- 3.0-3.2.6 *Honeycomb* (API level 11-13) - Lançada em 22 de Fevereiro de 2011, baseada no *kernel* Linux 2.6.36[14], a versão 3.0 *Honeycomb* foi a primeira feita exclusivamente para *tablets*, com *multitasking*[1] mais visual, com miniaturas dos aplicativos, adicionando a barra do sistema, que provê rápido acesso às notificações, e a barra de ação, que auxilia na interação com o aplicativo, disponibilizando diversas opções dependentes do aplicativo. Trouxe também suporte a processadores de vários núcleos e otimização de gráficos 2D e 3D, entre outros.

- 4.0-4.0.4 *Ice Cream Sandwich* (API level 14-15) - Lançada em 19 de Outubro de 2011, baseada no *kernel* Linux 3.0.1[15], trouxe nova interface com o usuário (UI), com animações mais refinadas. Foram incorporadas mais ações sem a necessidade de desbloqueio da tela, como o acesso direto à camera ou à barra de notificações, novas funcionalidades para a câmera, proporcionadas por APIs como as de detecção de face e de auto foco contínuo, suporte para *wi-fi*, P2P e outros recursos. Estabelecendo também a unificação da *API framework*, vários recursos e funcionalidades da versão *Honeycomb* foram incorporados, podendo ser utilizados também no desenvolvimento de aplicativos para celulares, assim o mesmo aplicativo pode ser desenvolvido e utilizado em celulares e *tablets* que rodem a mesma versão do Android (Android 4.0) ou superior. A versão 4.0.3 foi lançada posteriormente com a correção de *bugs* e algumas mudanças em APIs[16].

[1] Execução simultânea de mais de uma tarefa.

- 4.1-4.3 *Jelly Bean* (API level 16-18) - Lançada em 27 de Junho de 2012, baseada no *kernel* Linux 3.0.31[17], com a inclusão de recursos de otimização de desempenho, como *triple buffering*[2] e suporte para OpenGL ES 3.0[3], possibilitando o seu acesso por aplicativos através do *framework* ou de APIs nativas, em dispositivos com *hardware* gráficos que o suportam, o que torna a capacidade gráfica bem superior às das versões anteriores[18].

  Começando no Android 4.2, a ferramenta *Verify Apps* tornou-se disponível para os usuários, fazendo uma verificação de aplicações antes da instalação, notificando ou até mesmo bloqueando a instalação de aplicativos supostamente maliciosos. Na versão 4.3 é introduzido o SELinux no *kernel* Linux .

- 4.4 *KitKat* (API level 19) - Lançada em 31 de outubro de 2013[19], essa versão traz recursos úteis para o usuário final, como uma *framework* de impressão, um recurso de gravação de tela que permite gerar vídeos do que se passa na tela do dispositivo e melhorias na aceleração de GPU. Também há várias melhorias importantes na área da segurança, como a ativação do *enforcing mode*[4] no SELinux, a adição do *dm-verity*[5] e a separação das VPN's de cada usuário.

2.2 ARQUITETURA ANDROID

A plataforma Android foi desenvolvida para ser uma plataforma completa, oferecendo um *kernel* otimizado para dispositivos móveis, um *framework* provendo acesso a funções do sistema operacional e aplicações pré-instaladas. Ela pode ser subdividida em cinco camadas principais, ilustradas na figura 2.1[20], sendo elas:

1 *Kernel*

Baseado no *kernel* Linux 3.0[17], o núcleo do Android é otimizado para rodar em dispositivos com pouca memória e que necessitam de pouco consumo de energia. Ele é responsável pelas funcionalidades centrais do sistema, como gerenciamento de memória e de energia, gestão de processos e protocolos de rede. Nessa camada também podem ser incluídos os *drivers*, que interagem diretamente com o *kernel* e controlam o funcionamento do *hardware* do dispositivo, como câmera, tela e áudio.

[2]Uso de um terceiro buffer quando necessário.

[3]API gráfica descrita na seção 2.2

[4]Modo em que o SELinux bloqueia as operações que infrinjam alguma de suas regras. Discutido em detalhes na seção 3.1.2

[5]Ferramenta que verifica o sistema de arquivos em busca de alterações não permitidas. Discutida em detalhes na seção 3.2

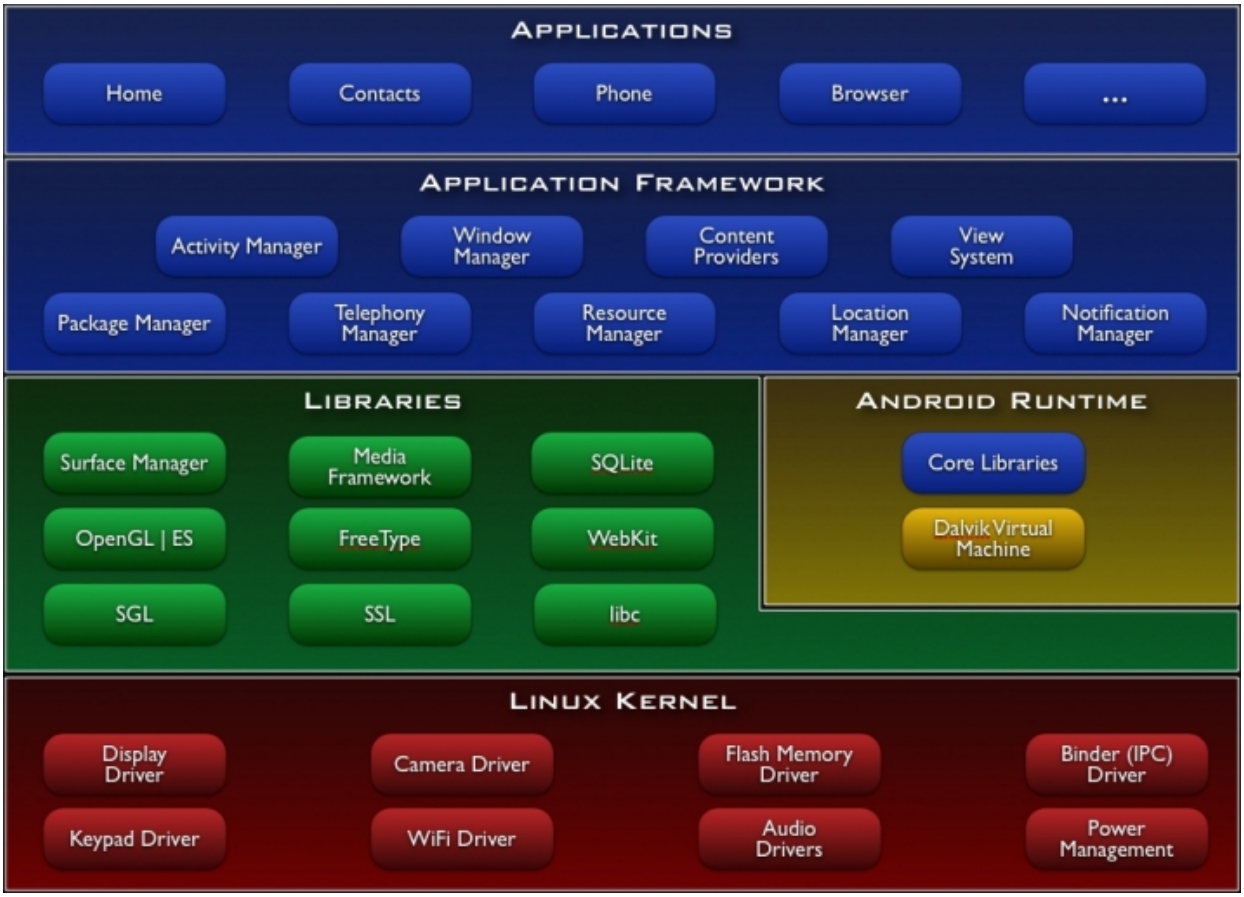

FIG. 2.1: Divisão da arquitetura Android em camadas[20]

2 Bibliotecas Nativas

São escritas em C/C++ e proveem funcionalidades básicas que podem ser acessadas por desenvolvedores através do *framework* de aplicação. Dentre estas bibliotecas estão:

- OpenGL ES: OpenGL é uma API gráfica multiplataforma usada para o processamento de gráficos 3D. OpenGL ES é uma versão do OpenGL voltada para dispositivos embarcados[21];
- WebKit: Motor de renderização de páginas *Web*, de código aberto, originalmente desenvolvido pela Apple Inc;
- SQLite: Gerenciador de banco de dados relacional. O Android provê métodos que permitem que os aplicativos gerenciem seus bancos de dados[22]; e
- *Framework* Multimídia: É escrita em C/C++, ao contrário das outras *frameworks* de aplicação, pois as operações realizadas pelos *codecs* exigem muito processamento e precisam ser otimizadas.

3 Android *Runtime*

Inclui a máquina virtual Dalvik, projetada para ser leve e apresentar pouca demanda por memória e processamento, e bibliotecas escritas em Java, que são executadas pela Dalvik.

4 *Frameworks* de Aplicação

São escritos em Java e proveem abstrações das bibliotecas nativas, ou seja, classes que atuam em conjunto para dar acesso aos serviços dessas bibliotecas. Cada um desses conjuntos de classes gerenciam uma determinada função do dispositivo. Essas funções incluem gerenciadores de atividades (*activities*), de telefonia e de localização, além de *content providers*, que propiciam a troca de informações entre aplicativos.

5 Aplicações

São os aplicativos com os quais os usuários interagem diretamente. São escritos em Java e cada um deles é executado em uma máquina virtual independente. O Android já traz alguns aplicativos pré-instalados, como navegador *Web*, gerenciador de contatos, tocador de mídia, entre outros.

2.3 APLICATIVOS ANDROID

Aplicativos Android são escritos na linguagem de programação Java. O SDK compila o código Java e traduz o Java *bytecode* para um arquivo *.dex*, *bytecode* próprio da máquina virtual do Android. Este *bytecode* é empacotado junto com quaisquer dados e *resource files*[6] em um *Android package*, um arquivo do tipo *.apk*. Todo o código em um único arquivo *.apk* é considerado uma aplicação, e é o arquivo que os dispositivos que utilizam Android usam para instalar a aplicação.

2.3.1 COMPONENTES DE UM APLICATIVO

Componentes de um aplicativo (ou aplicação), são as bases da estrutura do aplicativo Android. Estas são divididas em quatro grupos: *Activities*, *Services*, *BroadcastReceivers* e *ContentProviders*. Cada tipo de componente possui um papel específico e distinto na construção do aplicativo, possuindo ciclos de vida distintos, que indicam como a componente é criada e destruída.

[6]Arquivos do tipo imagens, áudio, entre outros.

Uma aplicação não precisa necessariamente possuir todos os quatro tipos de componentes, mas para apresentar uma inteface gráfica com o usuário é necessário possuir ao menos uma *activity*.

- *Activities*: É a camada de apresentação da aplicação. Uma *activity* representa uma única tela com uma interface com o usuário. Uma *activity* pode, por exemplo, mostrar a lista de contatos ou uma lista de itens do menu. *Activities* são independentes entre si mas podem se comunicar, e uma *activity* pode iniciar outra passando um *intent*, que será tratado posteriormente, para o método startActivity(), o que fará com que a nova *activity* seja mostrada.

- *Services*: São componentes que executam em *background*. São usados para realizar operações repetitivas e de longa duração ou em processos remotos e não possuem interface com o usuário. Outro componente, como uma *activity*, pode iniciar um *service*, e este continuará executando em *background* mesmo que o usuário alterne para outra aplicação. Além disso, um componente pode fazer um *bind* (ligação) com um *service* para interagir com ele.

  O *service* é iniciado quando um componente de aplicação, como uma *activity*, inicia-o chamando startService(). Geralmente, um serviço iniciado realiza uma única tarefa e não retorna o resultado para quem o chamou. Por exemplo, pode-se fazer o *download* de um arquivo em uma rede e, quando a operação termina, o serviço termina, sem a necessidade de interação do usuário.

  O *service* está ligado quando um componente de aplicação faz um *bind* com ele, chamando o método bindService(). O componente é capaz de interagir com o serviço ligado, enviar requisições, obter resultados e mesmo fazer comunicação entre processos. Um serviço *bound* (ligado) roda apenas enquanto o componente de aplicação estiver vinculado a ele. Múltiplos componentes podem ser ligados a um serviço ao mesmo tempo mas, quando todos eles se desligarem dele, o serviço será destruído.

- *Content Providers*: Um *content provider* gerencia um conjunto compartilhado de dados de aplicação, permitindo o compartilhamento entre aplicações. Através do *content provider*, várias aplicações podem ter acesso, e até modificar, caso o *content provider* permita, dados guardados no *file system* (um banco de dados SQLite), em servidores na Internet ou em qualquer outro local de armazenamento que a sua aplicação tenha acesso. Por exemplo, o Android possui um *content provider*

que gerencia as informações dos contatos do usuário, dessa forma, possibilitando qualquer aplicação, com as devidas permissões, ler e escrever informações sobre alguma pessoa.

Um *content provider* é implementado como subclasse de *android.content.ContentProvider*, e deve implementar um conjunto padrão de APIs que permitam outras aplicações realizar certas operações.

- *Broadcast Receivers*: São componentes que, assim como *services*, realizam operações em *background*, porém são utilizados apenas para realização de operações de curta duração. Componentes desse tipo recebem e respondem a *broadcasts*. *Broadcasts* são mensagens enviadas para todo o sistema contendo *intents*. Um *broadcast* do sistema pode ser um anúncio de que a bateria está baixa, de que a memória de armazenamento está acabando, entre outros, sendo possível o envio de *broadcasts* também por aplicações.

  Mesmo não mostrando uma interface com o usuário, os *broadcast receivers* podem disparar várias ações quando um evento acontece, podendo criar uma notificação na barra de *status*, iniciar uma *activity* ou um *service* qualquer. Uma mesma aplicação pode ter vários *broadcast receivers*.

2.3.2 ANDROID MANIFEST

O arquivo *Android Manifes.xml* é um arquivo que deve estar na pasta raiz do projeto da aplicação e que contém declarações dos componentes da aplicação, os recursos de *hardware* e *software* usados, permissões e o menor nível de API requerido para rodar a aplicação, assim como as bibliotecas usadas entre outras informações.

2.3.3 INTENTS E FILTROS INTENT

Activities, *services* e *broadcast receivers* são componentes de aplicação que podem ser ativados por uma mensagem assíncrona chamada *intent*. *Intents* fazem as requisições de determinadas ações a componentes, estes podendo pertencer a outras aplicações além da própria. O *Intent* carrega o nome da ação a ser realizada, no caso de *services* e *activities*, ou o nome do evento sendo anunciado, no caso de *broadcast receivers*.

Intents podem ser filtrados por uma aplicação para especificar quais *intents* podem ser processados pelos componentes da aplicação. A lista de filtros é definida no arquivo

manifest, assim o Android pode determinar quais *intents* são permitidos antes de iniciar a aplicação.

2.4 DALVIK VM

A maioria dos aplicativos Android são escritos em Java e executam em uma máquina virtual chamada *Dalvik virtual machine*. Esta máquina virtual é especializada em dispositivos com poucos recursos disponíveis e foi desenvolvida especialmente para esse propósito, executando o seu próprio *bytecode*, o que a torna incompatível com a máquina virtual Java. Esse *bytecode* é gerado a partir do Java *bytecode* e é normalmente mais compacto que este. Cada aplicação roda em sua própria máquina virtual, o que exige que a Dalvik seja muito leve, pois várias das suas instâncias estarão em execução simultaneamente.

As aplicações escritas para Android são compiladas como qualquer outra em Java, muitas vezes utilizando as mesmas ferramentas. Porém, em vez das classes serem comprimidas e empacotadas em um arquivo '.jar', elas são traduzidas para um arquivo '.dex', que incluem os *bytecodes* a serem executados pela máquina virtual. Dalvik foi projetada para reduzir o número de operações de leitura/escrita na memória, para isso utiliza um conjunto de instruções onde cada instrução tem um tamanho fixo de 16 *bits* além de registradores de tamanho 32 *bits*[23].

Logo após a instalação, todo aplicativo é verificado por uma ferramenta chamada *dexopt* para checagem de sua integridade. Para isso, a *dexopt* carrega todas as classes numa máquina virtual e roda todas as instruções em busca de operações ilegais, por exemplo, invocar um método com escopo de pacote a partir de uma classe num pacote diferente[24]. Classes que tenham sido verificadas com sucesso recebem uma *flag* indicando isso. Após confirmada a integridade, um *checksum* é gerado e guardado junto com cada arquivo '.dex'. Desse modo, cada vez que o aplicativo for executado futuramente, apenas o *checksum* será verificado. O *dexopt* também realiza uma otimização do código visando diminuição do tamanho e da complexidade do mesmo.

2.4.1 ZYGOTE

A máquina virtual Dalvik dispõe de alguns mecanismos para otimização do uso de memória. Entre eles cabe ressaltar o Zygote, que é um processo e inicializa classes que serão usadas frequentemente por várias aplicações. Desse modo, minimiza-se o consumo de memória, pois cada uma dessas classes é carregada apenas uma vez em vez de ser carregada uma vez para cada máquina virtual que irá utilizá-la. A Dalvik também provê

um coletor de lixo (*garbage collector*) que roda independentemente em cada instância da máquina virtual.

2.4.2 JIT

Nas primeiras versões do Android, Dalvik era simplesmente um interpretador sem a capacidade de realizar compilação *Just in Time*. Apenas a partir da versão 2.2 essa funcionalidade foi adicionada para processadores ARM[23], e a partir da versão 4.2 do Android já há o suporte de compilação JIT para a arquitetura x86[25].

O compilador JIT do Android utiliza um método no qual detecta-se quais porções do código são executadas mais frequentemente durante o programa, e apenas essas porções serão compiladas e executadas como código nativo. Desse modo há uma economia de memória pois não há necessidade de compilar todo o código. Esse método também é utilizado pela Java Virtual Machine[26].

2.5 KERNEL

A versão atual do Android, *Jelly Bean*, utiliza um *kernel* baseado na versão 3.0 do *kernel* Linux[17], com modificações para atender a algumas necessidades especias da plataforma móvel e drivers fornecendo funcionalidades adicionais. As capacidades de gerenciamento de energia do *kernel* são especialmente importantes para os dispositivos móveis.

Por conta das modificações só presentes no núcleo do Android, os desenvolvedores da plataforma têm dificuldade em acompanhar o ritmo de desenvolvimento do *kernel* principal, que já está na versão 3.11[27], pois inserir novamente todas as modificações em cada versão demanda muito trabalho. Para resolver esse problema, no final de 2011 foi iniciado um projeto[28] que visa gradualmente inserir no núcleo principal as modificações feitas especialmente para o Android. O projeto tem três objetivos principais[28]:

a) Permitir que um desenvolvedor utilize a última versão do *kernel* Linux para rodar um sistema Android sem que modificações sejam necessárias.

b) Permitir que sejam desenvolvidos *drivers* que funcionem nos dois *kernels* com pouca ou nenhuma modificação.

c) Diminuir o fardo causado pela migração das modificações de versão para versão.

O projeto está portando para o *kernel* principal alguns recursos importantes que antes só estavam incluídos no Android. Entre os mais notáveis estão[29]:

1 *Wake Locks*: Permitem que aplicações rodando na Dalvik VM impeçam que o sistema entre em estado de suspensão ou desligue a tela.

2 ASHMEN: A memória compartilhada anônima é um bloco de memória que pode ser usado por vários processos simultaneamente. Porém, diferentemente da memória compartilhada normal, ela pode ser liberada pelo *kernel* caso necessário.

3 *Binder*: O mecanismo de comunicação inter-processo do Android.

4 *Low memory killer*: Mecanismo que classifica os processos de acordo com sua importância para ser capaz de terminar processos pouco importantes em caso de falta de memória.

3 MECANISMOS DE SEGURANÇA NO ANDROID

A segurança no Android foi construída tendo em vista três objetivos principais[30], que são: proteger os dados do usuário, proteger os recursos do sistema e prover isolamento entre as aplicações. Para atingir esses objetivos foram incluídos no Android alguns mecanismos de segurança, que serão descritos a seguir.

3.1 CONTROLE DE ACESSO

Em geral, mecanismos de controle de acesso visam garantir que um usuário ou um processo só possa acessar recursos ou arquivos aos quais ele tenha permissão. Existem vários níveis de permissão de acesso. A seguir serão explicados os dois mecanismos de controle de acesso presentes no Android.

3.1.1 CONTROLE DE ACESSO DISCRICIONÁRIO (*SANDBOX*)

O Kernel Linux provê ao Android um modelo de permissões baseado em ID's de usuário (UID) e ID's de grupo (GID). Em sistemas Linux tradicionais, cada usuário do sistema é associado a um UID e pode ser associado a um ou mais GID's. Um arquivo no sistema pertence a exatamente um usuário e um grupo, e são definidos três conjuntos de permissões: as do dono do arquivo, as dos usuários que pertencem ao grupo do arquivo e as de qualquer outro usuário. As permissões possíveis são de leitura, escrita e execução.

No Android, os UID's e GID's são associados a aplicativos, em vez de usuários. Cada aplicativo é rodado num processo que tem as permissões de seu UID, desse modo cada aplicativo só pode acessar seus próprios arquivos, os arquivos de seu grupo ou os arquivos que podem ser acessados por todos. Isso garante o isolamento entre os aplicativos a nível de *kernel*, comumente chamado de *Application Sandbox*. Apesar de, por padrão, cada aplicativo só ter acesso a seus arquivos, o desenvolvedor pode explicitamente permitir que os arquivos de sua aplicação sejam acessados por outras aplicações.

Por estar a nível de *kernel*, a *sandbox* se aplica a todo processo rodando acima do *kernel*, o que inclui as bibliotecas nativas, as *frameworks*, a Dalvik e os aplicativos em geral.

3.1.2 CONTROLE DE ACESSO OBRIGATÓRIO (SELINUX)

O controle de acesso obrigatório no Android é provido pelo módulo do kernel chamado SELinux, que foi desenvolvido pela Agência de Segurança Americana, NSA. Ele funciona por meio de *labels* e de regras que as controlam. Foi introduzido na versão 4.3 do Android, porém, nessa versão ele roda no modo *permissive*[31], que apenas loga tentativas de violar as regras, mas não as impede. A partir da versão 4.4 o SELinux passou a ser executado no modo *enforcement*[32], que bloqueia as tentativas de violar as regras.

Cada arquivo e diretório tem uma *label*, assim como cada processo, porta de comunicação e dispositivo. As regras são escritas para controlar as ações que um processo que tem uma determinada *label* pode tomar em relação a um objeto com outra *label*, um arquivo, por exemplo. Um conjunto de tais regras é chamado de *policy*. É função do administrador do sistema definir as regras contidas nas *policies* ativas em seu sistema, porém, a maioria dos sistemas com SELinux já vem com um conjunto padrão de regras definidas.

O SELinux dispõe de vários modelos para a atribuição de *labels*, que serão descritos a seguir:

1 *Type enforcement*

Nesse modelo, cada processo recebe uma *label* com base em seu tipo, e cada objeto também recebe uma *label* baseada em seu tipo. Por exemplo, num sistema Linux tradicional com SELinux, os processos do Apache recebem a *label* $httpd\_t$ e os arquivos utilizados pelo Apache recebem as *labels* $httpd\_sys\_content\_t$ (para leitura) e $httpd\_sys\_content\_rw\_t$ (para escrita). Se o processo do Apache fosse *hackeado*, mesmo que ele estivesse rodando com permissões de superusuário, ele só conseguiria ler os objetos que tivessem a *label* $httpd\_sys\_content\_t$ e só conseguiria alterar objetos com a *label* $httpd\_sys\_content\_rw\_t$[33].

2 *MCS Enforcement*

Esse modelo é usado para os casos em que se deseja separar processos de mesmo tipo. Ele funciona da seguinte maneira: caso tenha-se dois processos, digamos proc1 e proc2, de tipo $proc\_t$, e dois arquivos file1 e file2, de tipo $file\_t$, e deseja-se que apenas proc1 possa acessar file1 e apenas proc2 possa acessar file2, então adiciona-se uma nova seção na *label* de cada um deles. Por exemplo, a *label* de proc1 viraria *proc_t:p1* e a de proc2, *proc_t:p2*, e a *label* de file1 seria *file_t:p1*, e a de file2, *file_t:p2*. Desse modo, além de

verificar a *label* de tipo, o SELinux verifica também a *label* adicional, chamada de *MCS label*, e garante o acesso apenas se as duas coincidirem.

3 *MLS Enforcement*

Esse modelo é semelhante ao MCS, porém menos utilizado[33]. Ele também funciona por meio da adição de uma nova seção nas *labels*, porém as labels MLS não precisam coincidir para que o acesso seja garantido. No *MLS Enforcement* é inserido o conceito de dominância entre *labels*. Retomando o exemplo do servidor Apache, poderia-se rodar dois servidores Apache, um com a *label httpd_t:secreto* e o outro com *httpd_t:ultra-secreto*. O primeiro só seria capaz de acessar arquivos com a label *httpd_sys_content_t:secreto*, enquanto o segundo poderia acessar tanto estes quanto os que tivessem a *label http_sys_content_t:ultra-secreto*.

3.2 VERIFICAÇÃO DO SISTEMA DE ARQUIVOS

A partir da versão 4.4, o Android oferece em seu *kernel* uma funcionalidade chamada *dm-verity*. Ela ainda está em fase experimental e é opcional, devendo ser ativada no momento de compilar o *kernel*.

O objetivo dessa ferramenta é impedir que *rootkits* modifiquem o sistema de arquivos e se instalem permanentemente no dispositivo. Para isso, ele verifica os blocos do sistema de arquivos em busca de modificações. A cada bloco do dispositivo de armazenamento é associado um *hash* SHA-256. Uma árvore de *hashes* é formada, de modo que apenas o *hash* raiz da árvore precisa ser confiável para que todo o resto da árvore seja verificada. Caso algum bloco seja modificado, a mudança em seu *hash* será detectada. Uma chave pública é incluída na partição de *boot* para verificar a autenticidade do *hash* raiz.

3.3 SISTEMA DE PERMISSÕES

O sistema Android limita o acesso dos aplicativos a recursos do sistema potencialmente perigosos. Essa limitação é implementada em várias formas diferentes. Alguns funcionalidades são restringidas por uma falta intencional de API's para manipulá-las, por exemplo, não existe API para prover acesso direto ao cartão SIM. Existem certas API's, no entanto, que são capazes de acessar informações sensíveis. Entra elas estão:

- Câmera;
- Dados de localização;

- Acesso ao Bluetooth;
- Telefonia;
- Envio e leitura de SMS/MMS; e
- Conectividade de rede.

O acesso a essas API's sensíveis é controlado por meio do Sistema de Permissões do Android. Para utilizar estas API's, o aplicativo deve explicitar em seu arquivo *manifest* de que recursos ele vai precisar. No momento da instalação, o sistema informa ao usuário quais permissões aquele aplicativo está requisitando, como mostra a figura 3.1, e o usuário garante as permissões ou cancela a instalação. Não é possível garantir apenas uma parte das permissões requisitadas, ou o usuário garante todas, ou não será possível instalar o aplicativo.

FIG. 3.1: Tela de permissões ao instalar o aplicativo Firefox

O Google chegou a incluir, na versão 4.3 do Android, uma ferramenta chamada App Ops que permitiria controlar individualmente as permissões de cada aplicativo[34], de modo que seria possível usar um aplicativo sem ter que conceder todas as permissões que ele pedia. Porém, na versão 4.4.2 essa funcionalidade foi escondida[35], e engenheiros da Google comentaram que ela tinha sido lançada acidentalmente, pois ela foi desenvolvida apenas para uso interno[36]. No entanto, ainda é possível utilizar o App Ops caso se tenha um dispositivo *rooteado*, baixando aplicativos desenvolvidos por terceiros para esse fim[36].

3.4 ASSINATURA DE APLICATIVOS

A assinatura de aplicativos serve para identificar o autor de um aplicativo. Cada desenvolvedor deve assinar seu aplicativo com uma chave privada. Um certificado é gerado a partir dessa chave privada e guardado no pacote APK. Esse certificado é usado ao associar um UID à aplicação.

Ao instalar um novo APK, caso o certificado deste seja assinado com a mesma chave privada de outro aplicativo no sistema, o novo APK tem a opção de especificar em seu *manifest* que ele quer compartilhar o mesmo UID com este aplicativo já existente[30]. Se isso for feito, os dois aplicativos poderão oferecer serviços entre si e trabalhar juntos por meio de *intents*. Na prática isso significa que um aplicativo poderá usar as permissões concedidas ao outro que tem o mesmo UID, mesmo que estas permissões não lhe tenham sido garantidas[37].

3.5 VERIFICAÇÃO DE APLICATIVOS

Existe uma ferramenta no Android chamada *Verify Apps*, que surgiu no Android 4.2, depois foi extendida para todas as versões acima da 2.3[38]. Ela é responsável por verificar cada aplicativo antes de sua instalação, em busca de código potencialmente perigoso, e avisar o usuário caso algo suspeito seja encontrado. Caso uma aplicação seja considerada realmente perigosa, sua instalação é automaticamente bloqueada[30]. Já há planos para melhorar o *Verify Apps*, para que ele seja capaz de monitorar os aplicativos continuamente, e não apenas na hora da instalação[38].

O *Verify Apps* conta com o auxílio de outra ferramenta chamada *Bouncer*. Esta, porém, não atua no dispositivo do usuário, e sim nos servidores da loja de aplicativos da Google, escaneando os aplicativos em busca de ameaças conhecidas e comportamento suspeito.

4 VULNERABILIDADES DA PLATAFORMA

O modelo de segurança da Plataforma Android possui algumas falhas de segurança, deixando o sistema suscetível a ataques. Uma vulnerabilidade na segurança do Android é algo que pode assumir amplos significados, podendo caracterizar falhas de segurança causada pela negligência na programação de aplicativos, na concepção de algumas das próprias API's desenvolvidas pela Google, até casos cuja maior parte do fator de insegurança advém do usuário. A possibilidade de instalação de aplicativos de terceiros (de fora da loja Google Play) pelo usuário, por exemplo, é em si uma vulnerabilidade, visto que é possível a utilização de lojas que não são autorizadas a disponibilizar aplicativos, e portanto sem a garantia de segurança da loja oficial, facilitando a manifestação de *malwares*.

Houve um caso de falha de segurança em que o serviço *Bluetooth* (com/android/phone/BluetoothHeadsetService.java), nas versões anteriores à 2.3.6 da plataforma, permitia ao atacante a possibilidade de obter dados por meio de ataques realizados via comandos AT[7][39]. Segundo o Android, esta vulnerabilidade foi resolvida a partir da versão 2.3.6[40].

Esta funcionalidade é muito utilizada por *malwares* para o próprio espalhamento, como o caso do *Backdoor.AndroidOS.Obad.a*, um complexo *trojan* descoberto pelo *Kapersky Lab* na metade de 2013, que mandava mensagens SMS, baixava outros *malwares* e infectava outros *smartphones* pelo *Bluetooth*. Depois de receber um comando de um servidor operado por um criminoso, o *malware* procura ao seu redor por aparelhos com conexões *Bluetooth* abertas e tenta enviar um aplicativo malicioso para quem encontrar.

Entre os principais transtornos que os aplicativos maliciosos causam estão o furto de senhas, contatos, código IMEI e informações pessoais, casos muitas vezes em que o usuário é enganado por aplicativos falsos de bancos ou emails, o envio de SMS de custo elevado, spams e propagandas. A seguir são descritos algumas das falhas do sistema e ataques cometidos contra a plataforma.

[7]Comando AT é um comando usado para agendar um outro comando, script ou programa para executar em uma data e hora específica.

4.1 API'S INSEGURAS

Algumas API's do Android possuem falhas de segurança, o que causa vulnerabilidades grandes na plataforma. API's pouco documentadas e desenvolvidas sem uma visão voltada pra segurança podem gerar casos como o de 2011, em que um estudo[41] realizado por pesquisadores da Universidade de Ulm, na Alemanha, descobriu uma falha no mecanismo de autenticação ClientLogin, usado por muitos aplicativos Android.

O mecanismo utilizava um token de autenticação que era passado para o Google services, habilitando o acesso às contas dos usuários. Foram realizados testes em várias aplicações da Google, como o Calendário, Picasa e Contatos, analisando posteriormente os dados que essas aplicações enviavam via wireless através das suas API's. Após análise desse tráfego, chegaram à conclusão de que muitos dos processos de autenticação dessas aplicações não encriptam os dados do utilizador, por exemplo via SSL em HTTPS, enviando em texto puro essas credenciais, o que em redes públicas que estejam a ser monitoradas por *crackers* podia causar uma interceptação das informações do usuário por pessoas mal-intencionadas. Somente no Android 2.3.4 e posteriores que o HTTPS começou a ser utilizado na sincronização do Calendário e Contatos, sendo que o aplicativo Picasa ainda passou algum tempo depois utilizando apenas HTTP na sincronização[42].

Outro exemplo é a falha que foi detectada no início de 2013, em um estudo[43] por discentes da Universidade de Hannover, na Alemanha. Os pesquisadores descobriram que gerenciadores de senhas em *smartphones* Android não eram tão seguros como esperava-se. Analisaram alguns gerenciadores em um Nexus Galaxy rodando Android 4.0 e descobriram que a plataforma dificulta o trabalho para os programadores no desenvolvimento de gerenciadores de senhas realmente seguros.

Os *smartphones* são incapazes de trabalhar de forma otimizada com os navegadores, pois o Android não fornece uma API para integrar os gerenciadores com navegadores ou aplicativos e, por causa disso, os desenvolvedores utilizam uma solução alternativa insegura, que é usar a área de transferência do sistema operacional para entregar as credenciais de *login* para o navegador e aplicativos. Os usuários podem utilizar a área de transferência para copiar as credenciais de *login* dos administradores de senha e colá-los em aplicativos ou no navegador, mas isso representa um enorme problema, já que a área de transferência é um recurso global, que pode ser acessado por qualquer aplicativo sem necessidade de permissões específicas. Há ainda um problema maior, um serviço de notificação, presente no Android, que permite que aplicativos sejam notificados de todas as alterações de conteúdo realizadas na área de transferência:

`android.content.ClipboardManager.OnPrimaryClipChangedListener`

Isto pode ser explorado por malwares para interceptar todas as senhas transferidas através da área de transferência. Os pesquisadores ainda desenvolveram um programa de demonstração chamado PWSniff, que implementou precisamente essa funcionalidade. O PWSniff é executado como um processo em segundo plano e não requer quaisquer permissões. O programa realiza uma busca em todos os dados do *smartphone*. Os desenvolvedores de malware devem colocá-los na mesma combinação da área de transferência, para capturar as credenciais. Todos esses problemas relacionados a segurança das API's são quase que totalmente responsabilidade da equipe de desenvolvedores, que muitas vezes não programam de forma segura suas aplicações, e também da equipe da Google.

4.2 UTILIZAÇÃO DE ROOT

Muito embora a maioria dos usuários de *smartphones* desconheça ou não se interesse por tal processo, alguns costumam ficar tentados a realizar o *root* de seus aparelhos para terem mais liberdade, o que gera vulnerabilidades. O processo de *root* do Android consiste em permitir que os usuários tenham acesso de administrador ao sistema operacional do seu *smartphone*, ou seja, arquivos e configurações que antes eram restritos somente a processos de sistema, ficam livres para serem acessados pelo superusuário.

Embora o recurso, que é basicamente um desbloqueio total do sistema, possa trazer diversos benefícios, é importante lembrar que ele também representa riscos para o dispositivo. Se antes os arquivos fundamentais do sistema ficavam protegidos, agora eles podem ser deletados de forma acidental pelos próprios usuários e comprometerem todo o funcionamento da plataforma.

Deve-se tomar muito cuidado ao realizar esse processo, pois muitos métodos não são totalmente seguros. Um dos maiores atrativos em se fazer o *root* do Android é a possibilidade de instalar ROMs de terceiros, que muitas vezes são mais atualizadas e possuem mais recursos do que aquelas que acompanham o aparelho originalmente. No entanto, tais versões podem vir embarcadas com aplicativos ou códigos maliciosos.

Ferramentas como os aplicativos Kingo Root e vRoot, ambas utilizadas para fazer *root*, aparentemente servem uma gama de aparelhos, indo de Galaxys a HTCs, porém nota-se um funcionamento estranho destas. VRoot e KingoRoot compartilham grande parte do mesmo comportamento suspeito, com a recuperação de binários do *smartphone* para um servidor remoto. O Kingo Root, além de ser de uma empresa da qual se tem

pouca informação, copiava até o IMEI e número de série do aparelho, sem razão aparente, algo que, segundo os desenvolvedores, não ocorre mais. Os diversos dados copiados do seu aparelho são enviados para servidores na China. Esse envio de dados confidenciais não criptografados para servidores na China e mesmo o fato de pegar o IMEI sem razão aparente são fatores que reforçam a idéia de que se deve evitar esses aplicativos.

Tendo em vista os riscos da realização do *root* e as várias vulnerabilidades desse processo, é muito importante que o usuário se certifique que está instalando uma ROM ou aplicativo de fontes que sejam confiáveis, ou seja, aquelas que possuam boas recomendações e aceitação por um grande número de usuários.

4.3 ESCALADA DE PRIVILÉGIOS

A escalada de privilégios permite alguém a causar sérios danos ao sistema. Um agente mal intencionado burla o sistema para garantir mais privilégios para uma ação específica do que ele originalmente tinha e, no pior caso, conseguir privilégios de superusuário, ou seja, um usuário pode instalar um aplicativo e este pode tomar conta do dispositivo, podendo gerenciá-lo totalmente, já que este obteve permissões de uso de superusuário. Existem dois tipos de escalada de privilégios: a horizontal e a vertical.

4.3.1 ESCALADA DE PRIVILÉGIOS HORIZONTAL

Na escalada de privilégios horizontal, o agente mal intencionado ganha direitos de acesso possuídos por outras entidades que estão no mesmo nível que o seu. No Android, esse é o caso em que um aplicativo ganha o direito de ler arquivos de outro programa, ou consegue roubar permissões de outro programa, que ele não possuía anteriormente, como o acesso à localização do dispositivo ou o acesso à rede.

Em [44], foi descoberto um exemplo de escalada horizontal no Android. Ele ocorre quando um aplicativo requer permissões que não existem em uma versão mais antiga do Android. Ao ser instalado num dispositivo com essa versão, o aplicativo não consegue as permissões solicitadas, pois elas não existem, porém, se for feita uma atualização do sistema para uma versão que suporte aquelas permissões, o aplicativo as ganha automaticamente, e o usuário acaba não sendo avisado das permissões adicionais que foram garantidas. O aplicativo pode então, após a atualização do sistema, ter acesso a informações importantes do dispositivo ou do usuário, sem que o mesmo saiba.

4.3.2 ESCALADA DE PRIVILÉGIOS VERTICAL

Na escalada de privilégios vertical, o agente mal intencionado consegue adquirir prvilégios de uma categoria mais alta à que possuía. Para isso, normalmente ele tenta explorar uma falha em programas que rodem a nível de *kernel*, para roubar seus direitos de super-usuário.

4.4 APLICATIVOS COM EXCESSO DE PERMISSÕES

Aplicações com excessos de permissões são muito comuns hoje em dia, o que dificulta o reconhecimento de aplicativos maliciosos pois o excesso de permissões torna-se um quesito mais fraco para o reconhecimento de aplicativos maliciosos. Mas ainda as permissões são algo que sempre devem ser levadas em conta antes de instalar um aplicativo.

O game *Flappy Bird*, que obteve grande sucesso na Google Play Store, mas já foi retirado do ar pelo seu criador, foi alvo de muitas clonagens, e muitos malwares foram feitos para se fazer passar pelo jogo. A maioria dos aplicativos é direcionada ao Android e é bem parecida com o *Flappy Bird* original, chegando a ter nome e ícone idênticos. Contudo, ao serem instalados, requerem permissões diferentes, como acesso à SMS, cartão de memória e até mesmo à bateria (para evitar que o dispositivo entre em *stand by*). Na versão autêntica, era preciso apenas habilitar a conexão com a internet para que anúncios pudessem ser exibidos, além do *ranking*.

Em outro caso de programa malicioso, é preciso enviar um SMS para ter a versão completa do game, o que gera desconfiança pois o *Flappy Bird* era gratuito. Se o usuário recusar a pagar pela mensagem, o aplicativo é fechado, mas é possível notar que ele continua rodando em segundo plano.

Embora haja tantas pistas, para um usuário comum não é tão fácil notar os aplicativos falsos como o do exemplo, por isso deve-se prestar bastante atenção nas características destes antes de instalá-los. Só o excesso de permissões, por exemplo, não é o suficiente para garantir que o aplicativo não seja confiável, pois muitas vezes o desenvolvedor foi displiscente na programação do aplicativo e este ficou com mais permissões que o necessário.

Mesmo assim, a revisão consciente das permissões pedidas por uma aplicação é de vital importância na manutenção da segurança do sistema, dado que qualquer aplicação pode ter um impacto negativo para o utilizador, se possuir permissões para tal. O utilizador pode contar com a informação do *Android Market*, onde pode verificar o *feedback* presente nos comentários de outros utilizadores sobre uma dada aplicação. Dessa forma, pode ter

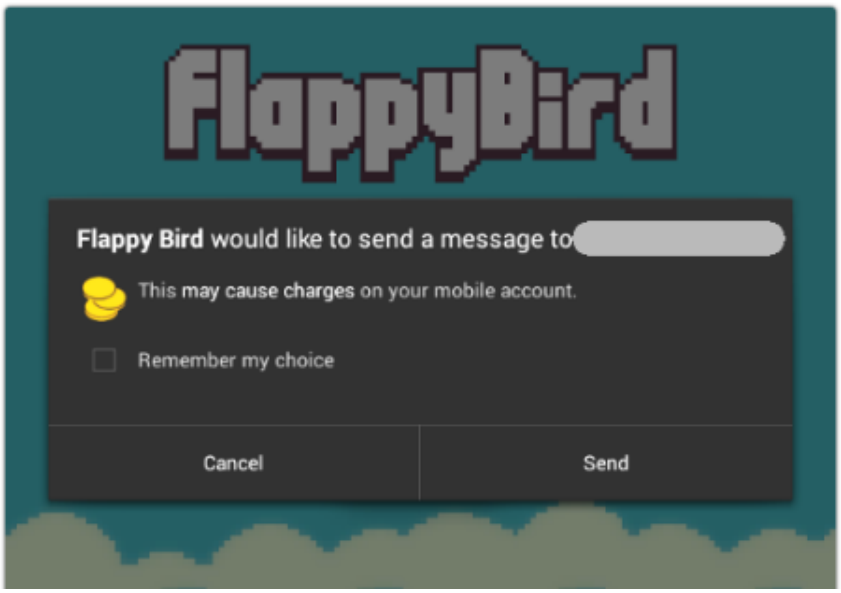

FIG. 4.1: *Malware* clone do *flappy bird* requisitando envio de SMS

uma maior consciência de outras opiniões e experiências quando pretende instalar uma aplicação.

4.5 ALTERAÇÕES DAS FABRICANTES

O atual modelo de negócios do Android provê às fabricantes uma grande liberdade para modificar o sistema antes de distribuí-lo em seus dispositivos. Apesar de todas serem baseadas na versão do AOSP (Android *Open Source Project*), são inseridas várias modificações, inclusive no *kernel*, com os mais diversos objetivos, sejam por questões de desempenho, segurança ou para tentar oferecer uma melhor experiência geral ao usuário.

Ocorre que nem sempre as empresas se preocupam o suficiente, ou têm uma equipe qualificada o suficiente para lidar com a segurança de seus sistemas ao realizar tais modificações, o que pode acabar inserindo vulnerabilidades de segurança que não existiam no Android original. Isso tornará mais difícil o diagnóstico do problema, visto que ele ocorrerá apenas nos dispositivos de uma fabricante específica.

Porém, do mesmo modo que uma empresa pode acidentalmente tornar seu sistema mais inseguro, ela também pode torná-lo mais seguro, caso tenha a devida preocupação com a segurança de seu sistema. No fim das contas, a fabricante acaba sendo uma variável importante a ser avaliada na hora de comprar um *smartphone* com Android.

5 MELHORIAS DE SEGURANÇA PROPOSTAS

Já foram apresentados nesse texto vários mecanismos que tem por objetivo tornar mais seguro o uso do sistema Android. Apesar de todos esses métodos terem sido implementados, ainda existem certos aspectos nos quais o Android poderia evoluir em termos de segurança. Serão apresentados a seguir alguns dos aspectos mais importantes, bem como fatores que influenciam negativamente a segurança da plataforma.

5.1 CONTROLE DE PERMISSÕES PELO USUÁRIO

Ao instalar um aplicativo, o usuário é apresentado a uma lista de permissões necessárias ao funcionamento do mesmo, e deve aceitar todas caso queira instalar. Seria interessante se fosse dada ao usuário a oportunidade de bloquear algumas das permissões pedidas pelos aplicativos, tanto no momento da instalação quanto após ela, caso desejasse.

É certo que a maioria dos usuários não iria usufruir dessa funcionalidade, porém essa é uma opção pedida por muitos dos usuários mais avançados, e com uma interface simples e explicativa mais usuários poderiam tirar proveito dela.

Essa é uma funcionalidade que já chegou a ser incluída na versão 4.3 do Android, porém foi removida na versão 4.4.2, sob o argumento de que foi desenvolvida apenas para uso interno pela Google, e que negar permissões aos aplicativos poderia levá-los a um mal funcionamento[35]. Porém bastaria que fossem criadas novas exceções no código Java para que os desenvolvedores pudessem tratar os casos em que certas permissões tenham sido negadas pelo usuário.

5.2 ATUALIZAÇÕES DIRETAS PARA O USUÁRIO

Existe hoje no mundo Android um grande fator de insegurança no modo como as atualizações chegam ao usuário final. Quando uma nova atualização é lançada pelo Google, ela não é disponibilizada imediatamente para *download* pelos usuários, mas sim enviada às fabricantes de dispositivos para que elas possam adaptar a atualização aos seus aparelhos (e a seus Androids, já que cada uma roda uma versão modificada do sistema).

O problema surge quando essas atualizações vêm para corrigir falhas de segurança críticas, pois, mesmo que a falha seja corrigida rapidamente pelo Google, a atualização

ainda deverá passar pelas fabricantes para que chegue ao usuário, e estas não têm sido tão rápidas na liberação de atualizações. Em muitos casos elas chegam a nunca lançar tais atualizações[45].

O ideal seria que a Google desenvolvesse um método que possibilitasse enviar as atualizações de segurança diretamente para os usuários, sem ter que depender das fabricantes, pois isso tornaria muito mais rápida a distribuição das correções. Essa, porém, não é uma tarefa fácil, pois devido à natureza *open source* do Android, a fabricante tem grande liberdade para alterar o código do sistema, e ficaria difícil para a Google saber se uma atualização entrará em conflito com o código alterado pelas várias fabricantes.

5.3 CRIAÇÃO DE PERFIS ISOLADOS

Existe um movimento atualmente sendo adotado por muitas empresas chamado BYOD (*bring your own device*), que visa estimular os funcionários a utilizar seus *smartphones* pessoais também no ambiente de trabalho, com permissões para acessar a rede interna e dados potencialmente críticos da empresa.

Isso pode apresentar vantagens como redução de custos para a empresa, pois ela não precisará fornecer *smartphones* para que seus funcionários utilizem no trabalho, e os funcionários não precisarão andar com dois celulares, pois um único já serviria para sua vida pessoal e profissional. Porém também leva a considerações de segurança, pois não há garantias de que os funcionários tomem medidas que garantam que seus *smartphones* estejam seguros o suficiente para serem usados na empresa. Os produtos da Apple têm levado vantagem em ambientes corporativos, pois eles dispõem de ferramentas para restringir e monitorar o dispositivo[46].

Uma solução para o Android seria a possibilidade de criar dois perfis separados, um para uso no trabalho e outro para uso pessoal. Esses perfis não poderiam compartilhar dados entre si, e cada um teria seus próprios aplicativos e contatos. A empresa poderia desenvolver ou adquirir soluções de terceiros para monitorar o celular do funcionário, mas isso não interferiria com sua privacidade, pois essas soluções seriam instaladas apenas no perfil profissional, e não teriam acesso a informações do outro perfil. Dessa forma a empresa estaria mais segura, pois possíveis brechas de segurança causadas pelo usuário só afetariam seu perfil pessoal, e ela poderia monitorar atividades suspeitas no perfil profissional dos celulares de seus funcionários sem interferir em sua privacidade.

Para garantir que o usuário não irá esquecer de trocar de perfil, poderia ser feito um monitoramento da posição geográfica, e no momento em que ele se aproximasse da

empresa o sistema automaticamente trocaria seu perfil para o profissional. Uma solução mais simples seria trocar para o perfil profissional no momento em que a rede da empresa fosse detectada pelo dispositivo, e voltar para o pessoal a partir do momento em que o sinal não fosse mais detectado.

6 COMPARAÇÃO COM OUTRAS PLATAFORMAS

Na atualidade, os sistemas operacionais dominantes no mercado mundial de *smartphones* são o Android, iOS e Windows Phone, com previsões de posições bem estáveis para os próximos anos[47]. Como são sistemas bastante diferentes, cada um com suas peculiaridades, a comparação entre as plataformas é algo bastante interessante e presente na nossa realidade, sendo constantes as discussões sobre qual o melhor sistema para se utilizar.

A Microsoft chegou bem depois de suas concorrentes nos *smartphones*, por isso ainda há pouca participação do Windows Phone nesse mercado, também possuindo pouca variedade de aplicativos na sua loja. Assim como o Android, essa plataforma é licenciável e está presente em diferentes modelos e marcas, mostrando visível crescimento.

Enquanto no caso do iOS temos uma integração do sistema operacional com o *hardware*, o que otimiza o sistema e gera ganhos em desempenho, para os dispositivos Android temos diferentes APIs dependendo de cada fabricante, pois cada empresa realiza as suas próprias modificações no sistema. As lojas da Apple e da Google podem ser equiparadas em termos de volume de aplicativos disponíveis.

O iOS tem atualização facilitada e amplamente divulgada pela Apple, algo pouco presente nos outros sistemas operacionais. No entanto, em todos os sistemas é preciso estar atento a versões e aparelhos compatíveis, pois alguns aparelhos podem não suportar as novas versões.

6.1 SEGURANÇA

Todos os três sistemas operacionais possuem o recurso de *sandbox*, com o isolamento de cada aplicativo em uso, sendo utilizadas áreas de armazenamento isoladas para a proteção contra acesso de outros aplicativos.

O sistema de permissões para os aplicativos também é semelhante no Windows Phone e no Android, sendo dada a permissão pelo usuário no momento da instalação e não sendo possível ampliação de permissões no momento da execução do aplicativo. No iOS, não há tal conceito de permissões na instalação, logo, não há como restringir as ações que um aplicativo pode tomar. Todos os aplicativos no iOS tem igual acesso a todos os recursos do sistema e, portanto, uma vez instalado no dispositivo, o aplicativo pode realizar qualquer

operação, apenas requisitando permissão para tal ao usuário no momento que for realizá-la. Assim é mais fácil gerenciar os recursos utilizados pelos aplicativos no iOS, pois o aplicativo necessariamente pede permissão sempre que vai utilizar um recurso do sistema.

É visível também a insegurança gerada pela abertura do Android para a instalação de aplicativos de fora da Google Store, em comparação com o iOS e o Windows phone, que só permitem a instalação de aplicativos das suas respectivas lojas oficiais.

A submissão de aplicativo na Windows Phone Marketplace é semelhante a do iOS na App Store, o desenvolvedor submete o aplicativo para revisão da Microsoft, que testa e coloca ou não na loja, a diferença é que o aplicativo ainda passa por um processo de certificação[48] na Windows Store. Já no Android esse sistema é mais livre.

Todos os sistemas operacionais tem diferentes pontos positivos e negativos, mas possuem bastantes semelhanças também. Os usuários devem atentar portanto, independente da plataforma escolhida, para manter um comportamento visando sempre a sua segurança, não instalando aplicativos de fontes desconhecidas (se a sua plataforma for desbloqueada, no caso do iOS e Windows Phone).

6.2 ESTATÍSTICAS

As estatísticas dos sistemas operacionais nas vendas de *smartphones* do terceiro semestre de 2013 são mostradas na figura 6.1. Pode-se notar que o Android mantém a liderança e é seguido pelo iOS e Windows Phone, respectivamente.

Essas posições tem a tendência de se manter nos próximos anos, o que mostra que o Android vai continuar sendo o maior alvo de *crackers* por um bom tempo, e portanto, continuará merecendo maiores e mais aprofundados estudos quanto à segurança.

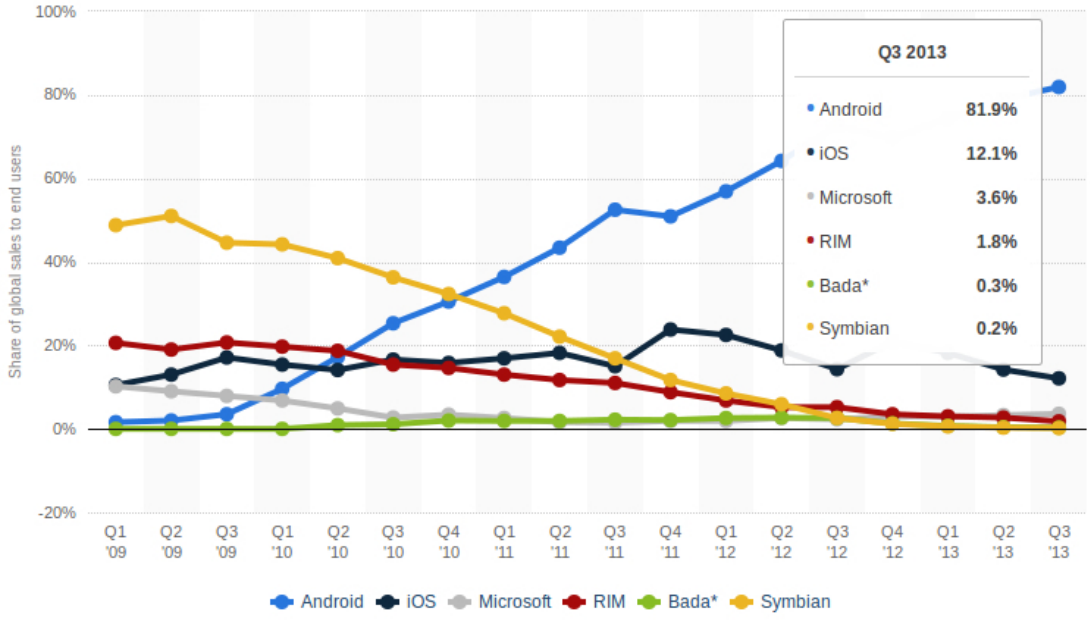

FIG. 6.1: *Market Share* dos sistemas operacionais móveis em vendas para o usuário final.[49]

7 CONCLUSÃO

No cenário atual da computação, no qual os ataques cibernéticos têm acontecido com frequência cada vez maior, a segurança dos sistemas deve ser levada a sério. Dado o grande crescimento do uso dos dispositivos móveis, o cuidado com a segurança destes deve ser ainda maior, visto que a tendência mundial é que as pessoas migrem cada vez mais suas atividades para esses dispositivos.

Este trabalho teve por objetivo fazer uma análise da segurança no Android. Foi feita uma introdução para contextualizar a situação atual da plataforma no que tange à segurança e para justificar a importância desse assunto no campo tecnológico. Dedicou-se um capítulo à explicação de como os vários elementos integrantes do sistema operacional funcionam e interagem entre si, com vistas a construir uma base para os capítulos seguintes.

Posteriormente, fez-se uma descrição detalhada dos mecanismos de segurança presentes na plataforma, bem como de métodos de ataque comumente utilizados contra sistemas móveis e vulnerabilidades que foram exploradas no passado. Além disso, foram sugeridas medidas que, se adotadas, poderiam melhorar o quadro geral da segurança no Android, e foi feita uma comparação com outros sistemas móveis no campo da segurança.

Existem algumas possibilidades de extensão futura do presente trabalho. Uma delas seria fazer uma pesquisa da segurança do Android rodando em outros dispositivos, como televisões, computadores rodando a versão x86 do Android, ou mesmo *single board computers*. Poderia-se também extender para o campo experimental, testando a eficácia dos diferentes tipos de proteção do sistema contra certos tipos de ataque.

O presente estudo contribuiu para o entendimento do funcionamento geral da plataforma Android, bem como dos principais mecanismos que o Android emprega para tentar evitar que os dispositivos sofram ataques cibernéticos, além de ter explicitado vulnerabilidades ainda existentes no sistema e como o mesmo poderia ser melhorado em alguns aspectos.

8 REFERÊNCIAS BIBLIOGRÁFICAS

[1] REED, B. *Windows Phone shows signs of life while BlackBerry keeps crumbling.* Disponível em: <http://bgr.com/2013/08/07/windows-phone-blackberry-market-share/>. Acesso em: 29 de setembro de 2013.

[2] REED, B. *Apple's once-dominant tablet market share has collapsed.* Disponível em: <http://bgr.com/2013/08/05/ios-android-tablet-market-share-2/>. Acesso em: 29 de setembro de 2013.

[3] CISCO Annual Security Report Documents Unprecedented Growth of Advanced Attacks and Malicious Traffic. Disponível em: <http://newsroom.cisco.com/release/1310011/Cisco-Annual-Security-Report-Documents-Unprecedented-Growth-of-Advanced-Attacks-and-Malicious-Traffic>. Acesso em: 12 de março de 2014.

[4] ALLIANCE FAQ | Open Handset Alliance. Disponível em: <http://www.openhandsetalliance.com/oha_faq.html>. Acesso em: 28 de setembro de 2013.

[5] ANDROID Git Repositories. Disponível em: <https://android.googlesource.com/>. Acesso em: 29 de setembro de 2013.

[6] ELGIN, B. *Google Buys Android for Its Mobile Arsenal - Bloomberg Businessweek.* Disponível em: <http://www.webcitation.org/5wk7sIvVb>. Acesso em: Arquivado do original em 24 de fevereiro de 2008. Recuperado em 20 de fevereiro de 2012.

[7] INDUSTRY Leaders Announce Open Platform for Mobile Devices. Disponível em: <http://www.openhandsetalliance.com/press_110507.html>. Acesso em: 28 de setembro de 2013.

[8] GOOGLE and the Open Handset Alliance Announce Android Open Source Availability. Disponível em: <http://www.openhandsetalliance.com/press_102108.html>. Acesso em: 29 de setembro de 2013.

[9] ANDROID Kernel Versions - elinux.org. Disponível em: <http://elinux.org/Android_Kernel_Versions>. Acesso em: 29 de setembro de 2013.

[10] ANDROID 1.6 Platform Highlights. Disponível em: <http://developer.android.com/about/versions/android-1.6-highlights.html>. Acesso em: 29 de setembro de 2013.

[11] ANDROID 2.0, Release 1 | Android Developers. Disponível em: <http://developer.android.com/about/versions/android-2.0.html>. Acesso em: 01 de outubro de 2013.

[12] DUCROHET, X. *Android 2.2 and developers goodies*. Disponível em: <http://android-developers.blogspot.com.br/2010/05/android-22-and-developers-goodies.html>. Acesso em: 30 de setembro de 2013.

[13] GINGERBREAD | Android Developers. Disponível em: <http://developer.android.com/about/versions/android-2.3-highlights.html>. Acesso em: 01 de setembro de 2013.

[14] ANDROID 3.0 APIs | Android Developers. Disponível em: <http://developer.android.com/about/versions/android-3.0.html>. Acesso em: 30 de setembro de 2013.

[15] UDALL, A. *Ice Cream Sandwich Runs on Linux Kernel 3.0.1*. Disponível em: <http://fineoils.blogspot.com.br/2011/10/ice-cream-sandwich-runs-on-linux-kernel.html>. Acesso em: 01 de outubro de 2013.

[16] ANDROID 4.0 APIs | Android Developers. Disponível em: <http://developer.android.com/about/versions/android-4.0.html>. Acesso em: 28 de setembro de 2013.

[17] LARABEL, M. *Google Working On Android Based On Linux 3.8*. Disponível em: <http://www.phoronix.com/scan.php?page=news_item&px=MTMxMzc>. Acesso em: 30 de Setembro de 2013.

[18] JELLY Bean. Disponível em: <http://developer.android.com/about/versions/jelly-bean.html>. Acesso em: 28 de setembro de 2013.

[19] ANDROID 4.4 KitKat é lançado oficialmente. Disponível em: <http://canaltech.com.br/noticia/google/Android-44-KitKat-foi-oficialmente-lancado/>. Acesso em: 12 de março de 2014.

[20] SYSTEM Architecture. Disponível em: <http://developer.android.com/images/system-architecture.jpg>. Acesso em: 29 de setembro de 2013.

[21] OPENGL ES | Android Developers. Disponível em: <https://developer.android.com/guide/topics/graphics/opengl.html>. Acesso em: 02 de outubro de 2013.

[22] SQLITEDATABASE | Android Developers. Disponível em: <developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>. Acesso em: 02 de outubro de 2013.

[23] BRAHLER, S.

Analysis of the Android Architecture — Fakultat für Informatik, Karlsruhe Institute of Technology, Outubro 2010.

[24] DALVIK Optimization and Verification. Disponível em: <https://android.googlesource.com/platform/dalvik/+/master/docs/dexopt.html>. Acesso em: 25 de fevereito de 2014.

[25] JELLY Bean | Android Developers. Disponível em: <https://developer.android.com/about/versions/jelly-bean.html>. Acesso em: 30 de setembro de 2013.

[26] THE Java HotSpot Performance Engine Architecture. Disponível em: <http://www.oracle.com/technetwork/java/whitepaper-135217.html#hotspot>. Acesso em: 30 de setembro de 2013.

[27] THE Linux Kernel Archives. Disponível em: <https://www.kernel.org/>. Acesso em: 30 de Setembro de 2013.

[28] LARABEL, M. *A Real Effort To Mainline Android Changes In Linux Kernel.* Disponível em: <http://www.phoronix.com/scan.php?page=news_item&px=MTAzMTY>. Acesso em: 30 de Setembro de 2013.

[29] ANDROID Mainlining Project - elinux.org. Disponível em: <http://elinux.org/Android_Mainlining_Project>. Acesso em: 01 de Outubro de 2013.

[30] ANDROID Security Overview | Android Developers. Disponível em: <http://source.android.com/devices/tech/security/index.html>. Acesso em: 10 de março de 2014.

[31] SECURITY Enhancements in Android 4.3. Disponível em: <http://source.android.com/devices/tech/security/enhancements43.html>. Acesso em: 09 de março de 2014.

[32] SECURITY Enhancements in Android 4.4. Disponível em: <http://source.android.com/devices/tech/security/enhancements44.html>. Acesso em: 09 de março de 2014.

[33] YOUR visual how-to guide for SELinux policy enforcement. Disponível em: <http://opensource.com/business/13/11/selinux-policy-guide>. Acesso em: 09 de março de 2014.

[34] AWESOME Privacy Tools in Android 4.3+. Disponível em: <https://www.eff.org/deeplinks/2013/11/awesome-privacy-features-android-43>. Acesso em: 10 de março de 2014.

[35] GOOGLE Removes Vital Privacy Feature From Android, Claiming Its Release Was Accidental. Disponível em: <https://www.eff.org/deeplinks/2013/12/google-removes-vital-privacy-features-android-shortly-after-adding-them>. Acesso em: 10 de março de 2014.

[36] XPOSED Module Brings Back App Ops on Android 4.4.2 to Give Your Control of Your Application Permissions. Disponível em: <http://www.xda-developers.com/android/xposed-module-brings-back-app-ops-to-android-4-4-2-and-gives-your-control-of-your-application-permissions>. Acesso em: 10 de março de 2014.

[37] MAIER, J.

Enhanced Android Security to prevent Privilege Escalation — Fakultat für Informatik, Der Technischen Universität München, September 2013.

[38] HOW Google's Android security is about to get even smarter. Disponível em: <http://blogs.computerworld.com/android/23590/google-android-security>. Acesso em: 10 de março de 2014.

[39] BLUETOOTH Vulnerability. Disponível em: <http://www.cvedetails.com/cve/CVE-2011-4276/>. Acesso em: 11 de junho de 2014.

[40] CVE Details. The ultimate security vunerability database. Disponível em: <http://www.cvedetails.com/vulnerability-list/vendor_id-1224/product_id-19997/Google-Android.html>. Acesso em: 12 de março de 2014.

[41] CATCHING AuthTokens in the Wild The Insecurity of Google's ClientLogin Protocol. Disponível em: <http://www.uni-ulm.de/en/in/mi/staff/koenings/catching-authtokens.html>. Acesso em: 11 de junho de 2014.

[42] CATCHING AuthTokens in the Wild. The Insecurity of Google's ClientLogin Protocol. Disponível em: <http://www.uni-ulm.de/en/in/mi/staff/koenings/catching-authtokens.html>. Acesso em: 12 de março de 2014.

[43] FAHL, S. et al.

Hey, You, Get Off My Clipboard — Distributed Computing & Security Group, Leibniz University of Hannover, 2013.

[44] WEAKNESS in Android Update Service Puts All Devices at Risk for Privilege Escalation. Disponível em: <http://threatpost.com/weakness-in-android-update-service-puts-all-devices-at-risk-for-privilege-escalation/104906>. Acesso em: 11 de junho de 2014.

[45] ANDROID should embrace a Windows-style security update model. Disponível em: <http://www.zdnet.com/android-should-embrace-a-windows-style-security-update-model-7000018029/>. Acesso em: 21 de maio de 2014.

[46] SO, you want to adopt BYOD? Disponível em: <http://www.androidcentral.com/so-you-want-adopt-byod>. Acesso em: 21 de maio de 2014.

[47] DESPITE a Strong 2013, Worldwide Smartphone Growth Expected to Slow to Single Digits by 2017, According to IDC. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS24701614>. Acesso em: 21 de maio de 2014.

[48] APP certification requirements for Windows Phone. Disponível em: <http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184843(v=vs.105).aspx>. Acesso em: 11 de junho de 2014.

[49] GLOBAL Market Share Data 2014 for Smartphone Operating System. Disponível em: <http://sourcedigit.com/1913-smartphone-os-global-market-share-data-2014/>. Acesso em: 22 de Maio de 2014.